LEPREAU

An Informal Critique of 4.2BSD

Rob Gurwitz
BBN Laboratories

Introduction

The development of 4.2BSD UNIX† was an ambitious undertaking, both in the number of new facilities and major subsystems that were designed and in the amount of code produced. The amount of input and community discussion about these ideas was impressive. In general, considering the size of the task and that the problems 4.2 set out to solve were difficult and of long standing, the result is pretty good. However, as those of use who were involved knew at the time, a lot of what was being built was experimental in nature. No one was sure how well the mechanisms would work, how they would be received, or how well they would answer the problems they tried solve.

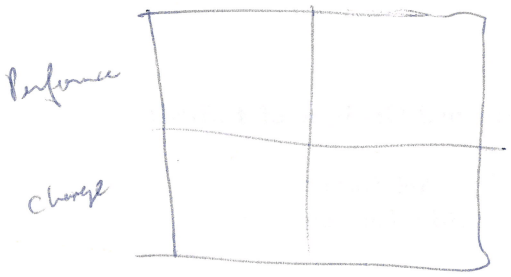
We've all had quite a while now to gain some experience with 4.2BSD and develop some reactions. This note contains a few of my own opinions. It centers on IPC and how 4.2 relates to distributed systems, since the IPC facilities seem to be the major new development in it over 4.1 and the thing that comes closest to a unifying abstraction in the system. Also, the trend toward distributing our computing resources is well upon us and appears to be the biggest demand to be met right now. What follows is a brief and somewhat random collection of topics that appear to be left pending in the wake of 4.2's introduction. They are not intended to be complete and are in no particular order. They are intended to spark discussion. The end of the note outlines a possible approach for future direction.

Some Problems With 4.2BSD

The additions made to UNIX in 4.2 have increased functionality in important ways (IPC, symbolic links, network services, non-blocking I/O, to name a few). This increase in functionality has not been without a price. First, the size of the kernel has expanded greatly with a concommittent number of new system calls. This happened because UNIX's structure required that many of these new facilities be added to the kernel for efficiency. Part of the original attraction of UNIX was its relatively simple and elegant structure as seen by the user, and the relatively compact and understandable implementation in the kernel as seen by the system implementor/maintainer. The increase in functionality has come at the expense of elegance, compactness, and understandability.

The division of function between kernel and user level can be somewhat arbitrary, but it seems that the trend set by 4.2 is excessive reliance on putting new functions in the kernel. There are several alternatives that could be pursued. One would be to examine the features currently in the kernel to see whether any are candidates for being put into user level libraries. A more fundamental change would be to examine mechanisms for moving functions out of the kernel into server processes, while retaining efficiency. "Lightweight processes," some form of shared memory IPC, and better access to the "raw" or link levels of network service are some desirable facilities for this that come to mind. (For example, it is

† UNIX is a trademark of AT&T Bell Laboratories


Performance
change

hard to implement new protocols outside the kernel in 4.2).

The second price has been a reduction in performance compared to 4.1, as documented by Leffler, Karels, and McKusick, and readily apparent to users of the old and new systems. Work is being done to remedy this, however, and one hopes this will not be a problem much longer, especially since many of 4.2's enhancements were done in the name of efficiency.

The facilities that were planned for 4.2, and documented in numerous revisions to the System Manual, are essentially incomplete. Mechanisms such as portals, virtual memory management enhancements for mapping files, process control, and terminal multiplexing improvement, remain unfinished, in various stages of design or implementation. There are some pretty glaring deficiencies. For example, the wide use of pseudo-teletypes in 4.2 network utilities for getting I/O to flow between the IPC system and the character I/O system represents a shortcoming, given the age and well-known kludginess of that mechanism.

Will these facilities ever be completed? More important, *should* they be completed in light of reaction to the size and sprawl of the 4.2 kernel?

4.2BSD and Distributed Systems

The relation between 4.2 and distributed systems is perhaps the most important area for new developments. It has been said many times that 4.2 is not a distributed operating system, but the implication has been that it provides the base for building a distributed environment. Given the state of the IPC facilities and the various "remote" utilities (*rlogin, rsh, rcp,* etc.), 4.2 seems to occupy some middle ground, neither fish nor fowl. As a base for building a distributed environment (new services or enhancements that move closer to the functionality of a distributed operating system), there are a number of unresolved issues that need to be addressed.

First, the IPC mechanism in 4.2BSD is too tightly bound to the semantics of the underlying virtual circuit network transport. Perhaps this is a natural consequence of the history of the development of the IPC and network components of the system and the backgrounds of the designers (myself included). It is evidenced by the fact that the socket constructs derive completely from traditional networking concepts and also that the virtual circuit transport protocol implementation (TCP/IP) and Internet addressing domain are the dominant, and really the only fully developed protocol implementations.

Naming in 4.2 IPC relies either on Internet addresses and TCP ports or the UNIX file name space for local operations. There is no attempt to provide any form of location independent global naming, even in the context of the local network (as opposed to the entire Internet). Thus, there are two disjoint name domains and no mechanism for bridging them. While it can be argued that a transparent naming system for local and remote access was not an intended part of the design, the use of a single mechanism for inter-machine and intra-machine IPC (sockets) leads one to wish that the design went further to provide consistent naming as well. For example, extending the use of string names to include the Internet as well as the UNIX domain would increase transparency. Another alternative would be the use of some other domain-independant naming structure for sockets, such as an additional mapping from "i-number" to Internet address.

- Sun will not do a D.O.S.
- Sun will

PYRAMID

1) Dir name defn
2) FIFO's
3) Signal handling
4) sys V IPC
5) tty driver + ioctl

FUTURE

1) VMEM
2) Tightly coupled
3) DFS

NOW Record locking (lint/group std) backward comp.
c/w forks
assign I/o
DFS (Record locking problem)

Tightly coupled -
- loosely based on Purdue's approach } incremental approach
- increasing symmetry
  - Swap

4.2 Enhancements - perfm.
- mbuf redesigned (more like swap area)
- libc.a optim.
  namei
  vmem/paging - algorithm
  Compile

want CPU to support!
- Smart DMA - mult. boundary
  scatter/gather
- Intelligent Periph. Processing
  - CPU for I/O requests
  - Peripheral handling of low level I/O processing
- Paging

A more serious problem with 4.2 IPC is access control and authentication. Here, the IPC mechanism provides no access control in and of itself, but depends heavily on the UNIX file protection mechanism in the UNIX domain, and the notion of "trusted ports" in the Internet domain. Not only is this an example of inconsistent semantics between inter- and intra-machine IPC, but the "trusted port" mechanism is far too weak: TCP port numbers can be easily spoofed. On the other hand, in the current system any process can post a *listen* on any "untrusted" port and any process is free to send to any other port. Also, access to the "raw" protocol layers is restricted to those with super-user privileges. This "all or nothing" approach to access is prevalent throughout the IPC mechanism. Some form of authentication service is needed to provide more effective protection.

The problem of heterogeneity is not dealt with by 4.2BSD, since it was designed with a homogeneous computing environment in mind. The current trend is toward a heterogeneous machine environment, including both VAXen and 68000 workstations. This environment requires that attention now be paid to data representation in some canonical, externally agreed-upon format. If Berkeley UNIX is to serve as a base for use in a wider, distributed environment, this problem must be dealt with.

These issues lead to a more generic problem related to the low level, network-oriented nature of the IPC mechanism. There is very little support provided for using the IPC facilities that do exist for developing distributed applications. Each application that is developed must deal with low level issues of addressing and naming, access control and authentication, external data representation, etc. This has resulted in the development of a few distributed utilities (the "remote" commands) that are *ad hoc*, restricted in functionality, and unsecure.

Whither Berkeley UNIX?

The criticisms given above naturally lead to the question of what to do about them and what to do next (if anything) with Berkeley UNIX. I think there is still a strong need for a single supported operating system that is available to the DARPA research community. UNIX has so far served the community fairly well in providing the services that it needs. Its wide acceptance both within the research community and, more generally, throughout the industry is evidence of its success. The challenge before us now is to evolve this system or choose another to meet the demands for distributed systems.

This is a result of the changing environment that 4.2 is being used in. Originally, Berkeley UNIX was designed as a set of enhancements to Version 7 for a particular machine (DEC VAX) in a mostly stand-alone environment. Sometime during the development of 4.2, this changed to a more distributed environment of homogeneous machines (still VAXen) on a high-speed local area network. Soon after its release, that environment changed yet again to include heterogeneous machines, VAXen and the crop of 68000-based workstations, most notably SUNs.

In order to determine how to evolve the system we must re-examine our goals. Do we want a full-blown distributed operating system with diverse functionality, including a transparent distributed file system, distributed process control, replication and migration facilities, and effective access control and authentication? If so, we may have to look beyond UNIX to provide these features in anything but an *ad hoc* manner.

Cal Steel

		VEG
12/82	V7M announced (~~announced~~)	6
8/83	Ultrix	8
8/83	V7M deliv.	30
11/83	Ultrix 32M	42
6/84	U-32 deliv.	50
9/84	Ultrix 32M	90
10/84	Ultrix II V2 deliv	95

Trans

- Stable work
  major release q 12 - 24 mos.
- WS emphasis, not cluster.
- Will support CI + HSC but not in a sophisticated way (no VMS "tricks")

K. Lantz — Ridge OS — provide Unix compatibility originally (Msg-based) + ~~RISC~~ RISC

DEMOS like kernel.

Kernel is 4K

release DFS middle '85 — NOT D.OS.

Committment to 100% fine-grained compatibility w/ 4.2 or SYS V

Although advertising says

will have shared memory + C/W

SUN

- * ~~dbx~~tool
  - fonttool
  - ttytool (> shelltool)

Graphics

- core
- GKS
- PHIGS
- VDI

DFS — stateless server

lose some unix semantics

- append mode (no state)
- file locking (flock)
- unlink open files — 3/4 solu -

File server — file middleware in the server node

Cache of authentication

lots of local caching

- no remote devices
- no remote "root" access

Assumes common uid/grid

"A "stateless file cache" is a performance enhancement"

My own opinion is that we want a compact, efficient, and elegant local operating system with a competent user interface. In addition, we need a way of integrating these hosts into local clusters based on high-speed LAN communication. This implies better low level communications support, including protocols for levels of network service other than TCP virtual circuits (e.g., sequenced packet or RDP, true datagrams, etc.), and higher level support for RPC or true, location independent IPC, and data representation and structuring facilities for hetergenous configurations. A transparent distributed file system, provision of a variety of other distributed services (naming, authentication, database access, etc.), and support for the development of new distributed applications are also requirements for this system. Finally, it should accommodate a variety of host classes (PCs, workstations, mainframes, and supercomputers) and hosts of different manufacturers and architectures.

UNIX, while far from perfect, meets many of the criteria for a good local operating system. I think the approach we must take is to tune our current UNIX system and extend it in less invasive ways to provide the distributed services we need.

An Approach for the Future

Given that set of goals, what should our path be? The first order of business is to complete those facilities the current 4.2 implementation has begun. That includes making the implementation faster, deciding which unimplemented features from the System Manual should be completed and which should be dropped. The goal should be to achieve a stable, well-tuned version of the system and build on that. New services should then be implemented at *user level* to achieve the goal of a UNIX-based distributed environment. Kernel changes should be minor and only done where absolutely necessary, though some attention should be paid to shrinking the kernel by moving functions to user level and offering better support for server creation. New services should be added using the IPC facilities and server processes.

The result should be made easily available to the entire community: universities, commercial developers, and manufacturers. By easily available, I mean sources at low-cost, with the least restrictive additional licensing possible. Future development should not be tied to a single vendor, but should be adaptable by vendors for their hardware.

Finally, it should be supportable and supported. I distinguish supportability from support here. Supportability means no gratuitous changes to the basic set of commands, better control of sources (kernel, commands, and libraries), and a separation of research work from the distributed system. Support means updates, including bug fixes. The system should at least be supportable. Real support probably can't be done under the current scheme. A consortium approach where a group of interested parties pool resources to pay for a central support organization might be one alternative.

— Admittedly, many of these goals are in conflict with the current realities of licensing terms, market realities, and the interests of commercial vendors. However, the past success of UNIX in the research community has come from easy source availability and an understandable, transportable system. Rashid has described the current trend toward fragmentation in 4.2 facilities the "Balkanization" of UNIX. I believe this fragmentation is also occurring in the marketplace. Unless we exercise some control over both the availability of common software and the underlying complexity of the system, we are in danger of losing a unique research vehicle: a widely available, easily understandable, transportable, common software base.

IRIS

1. Cut funded – non-negotiable needed 4.2 – porting 4.2 to IBM WS

50 Lisas from Apple – single planning cycle

Broadband net 95% BSDg.

Ed Gould: Bug list

4/10

Locus Questions?

protocol?
net access min?
FS perform + type?
Avail for user?
Network wide fsck
Unix passwd file?
TCP/IP?
15 Vax 750 100 users
1.5

Locus

Transparent Dist. Operations
- fs
- processes
- devices (including terminals) RAW devices only on local host.

Reliability & Availability
- replicated files
- Robust file store & software (Commit at lowest level is built in)
(+ full transactions)
- dynamic reconfiguration
- merge of replicated storage
- internal consistency checking

Unix compatibility
System V
Berkeley Unix
load modules

Problems
fs driver – rebuilt it
map signals – procs of different types

Heterogeneity
different cpus in single network
bridges to other systems (files + commands)

Performance
- equals Unix locally (3rd party std it – who?)
- far exceeds others remotely
- smaller system than 4.2 w/o UCB IPC + Fast filesystem

Net Work
- Locus over IP
- Sigpost ~~over~~ Locus
- IPC: 4.2, sysV. ~~for~~ handles are user supplied, UCB
- remote purging
- Transaction completion + integration
- Dist. record level locking
- R
- Replication revisited ↓ hard links

- Fast file sys
- Name table caching
- Scaling changes for multi-hop
more ~~local~~ node systems

Acknowledgement

Thanks to Dick Edmiston, Keith Lantz, Dave Mankins, Mike O'Brien, and Bob Walsh for their comments and ideas. Also, earlier critiques of 4.2 IPC by Lantz and Rick Rashid proved useful.

Better server support: - so can move stuff out of kernel

LW processes

Shared memory

Better link access (so can have protocol outs

For D.O.S.

DFS

RPC

Authent

Heterogr

- Availability
  - Low Cost sources
  - Least restrictive licensing
- Vendor independence
  (- stds.)
- supportable vs supported
  - Better source control
  - separate research contrib from released
  - consortium approach

RPC-based

455
NE

used fselect — no mem. of random contents

debugging problem

Have source version

internal → tuff →

Program
WM
User

jag: SUN & other WMs don't do right Pmg when window is resized.
Doesn't make life harder for applic, just more

(IBM sup will be able to be released later, including source.
License Model is Wisconsin TCP/IP IBM code - for academia)
WM - 80K T, 80K D start.

Brown, VCI, Plato at Illinois
(Bolsa)

A questionnaire to be published in J. of Higher Educ.
Student machines MAC } 2000 by end of acad '84
WS } family